OtoDecks DJ Application - Project Report

Author: Alan Nascimento
Date: December 2024
Course: Object-Oriented Programming Final Term Coursework

Table of Contents

R1. Program Functionality Description

Overview

OtoDecks is a professional DJ application built using the JUCE framework that simulates a dual-deck DJ system with advanced features for music mixing and beat synchronization. The application provides an intuitive interface for DJs to load, play, and mix music tracks with real-time waveform visualization and beat grid functionality.

Core Components

1. Main Application Architecture

The application follows a modular architecture with the following key components:

- **MainComponent**: Central orchestrator that manages the entire application
- **DJAudioPlayer**: Audio playback engine for each deck
- **CustomDeckControl**: Custom UI component for each deck with professional DJ controls
- **MusicLibrary**: Persistent music file management system
- **BeatGrid**: BPM detection and beat synchronization system
- **WaveformDisplay**: Real-time audio waveform visualization

2. Dual-Deck System

The application features two independent decks (Deck 1 and Deck 2) that can: - Load and play audio files simultaneously - Control individual volume levels - Adjust playback speed (pitch control) - Set cue points for precise track control - Synchronize with beat grid for professional mixing

3. Audio Processing Pipeline

Each deck implements a sophisticated audio processing chain:

```
Audio File → AudioFormatReader → TransportSource → ResamplingAudioSource → Out
```

Key features: - **Format Support**: Supports multiple audio formats (WAV, MP3, AIFF, etc.) - **Real-time Processing**: Low-latency audio playback with minimal buffer sizes - **Resampling**: Automatic sample rate conversion for compatibility - **Mixing**: Dual-deck audio mixing through MixerAudioSource

4. Professional DJ Controls

Each deck includes comprehensive controls: - **Transport Controls**: Play, Stop, Cue, Sync buttons - **Volume Control**: Master volume slider with real-time VU meter - **Speed Control**: Pitch/speed adjustment slider (0.5x to 2.0x) - **Position Control**: Track position slider for precise seeking - **EQ Controls**: Three-band equalizer (Low, Mid, High) - **Waveform Display**: Real-time audio visualization

---

R2. Music Library and Persistence Description

Library Management System

1. File Management

The MusicLibrary component provides comprehensive file management capabilities:

- **File Addition**: Users can add audio files through file browser or drag-and-drop
- **File Validation**: Automatic validation of supported audio formats
- **File Removal**: Individual file removal or bulk library clearing
- **File Organization**: Alphabetical sorting and display of track information

2. Persistence Implementation

The library implements robust persistence between application sessions:

```
// Library data file location
File documentsDir = File::getSpecialLocation(File::userDocumentsDirectory);
libraryDataFile = documentsDir.getChildFile("OtoDecksLibrary.txt");
```

Persistence Features: - **Automatic Saving**: Library state is automatically saved when files are added/removed - **Startup Loading**: Library is automatically restored when application starts - **File Path Storage**: Absolute file paths are stored for reliable file access - **Error Handling**: Graceful handling of missing or corrupted files

3. Data Structure

The library uses a vector-based data structure for efficient file management:

```
std::vector<File> libraryFiles;  // Main file storage
File libraryDataFile;            // Persistence file location
```

4. User Interface Integration

The library provides seamless integration with the main application: - **ListBox Display**: Clean, organized display of all library tracks - **Drag-and-Drop Support**: Direct file loading from external sources - **Track Selection**: Click-to-load functionality for immediate deck loading - **Visual Feedback**: Real-time updates and status indicators

5. File Format Support

The library validates and supports multiple audio formats: - WAV (Waveform Audio File Format) - MP3 (MPEG Audio Layer III) - AIFF (Audio Interchange File Format) - FLAC (Free Lossless Audio Codec) - OGG (Ogg Vorbis)

R3. UI Customization Description

Design Philosophy

The UI design follows professional DJ equipment aesthetics with a modern, dark theme inspired by high-end DJ controllers and software like Serato and Traktor.

Custom UI Components

1. CustomDeckControl Component

Each deck features a completely custom-designed interface:

Visual Design Elements: - Vinyl Disc Animation: Rotating vinyl disc visualization that responds to playback state - **Custom Color Schemes**: Different color schemes for each deck (dark gray for Deck 1, medium gray for Deck 2) - **Professional Layout**: Grid-based layout with proper spacing and alignment - **Animated Elements**: Rotating elements around the vinyl disc for visual appeal

Custom Graphics Implementation:

```
// Vinyl disc path creation
vinylDisc.addEllipse(0, 0, 100, 100);

// Rotating elements initialization
void initializeRotatingElements() {
    for (int i = 0; i < 8; ++i) {
        float angle = (i * MathConstants<float>::pi) / 4.0f;
        float x = 50.0f + 60.0f * cos(angle);
        float y = 50.0f + 60.0f * sin(angle);
        rotatingElements.push_back(Rectangle<float>(x-5, y-5, 10, 10));
```

```
    }
}
```

2. Custom Button Styling

All buttons feature custom styling with professional DJ aesthetics: - **Color-coded Functions**: Different colors for different button types - **Hover Effects**: Visual feedback on button interaction - **Professional Typography**: Bold, clear text labels - **Consistent Sizing**: Uniform button dimensions for professional appearance

3. Custom Slider Design

Sliders are customized for professional DJ use: - **Rotary Sliders**: BPM and EQ controls use rotary slider style - **Linear Sliders**: Volume and position controls use linear style - **Custom Colors**: Color-coded sliders for different functions - **Real-time Feedback**: Immediate visual response to parameter changes

4. Layout System

The application uses a sophisticated 3-column layout system:

```
// Custom layout proportions
int leftWidth = totalWidth * 25 / 100;    // 25% for library
int middleWidth = totalWidth * 30 / 100;  // 30% for beat grid
int rightWidth = totalWidth * 45 / 100;   // 45% for decks
```

Layout Features: - **Responsive Design**: Adapts to different window sizes - **Professional Proportions**: Optimized space allocation for each component - **Visual Hierarchy**: Clear separation between functional areas - **Consistent Spacing**: Uniform margins and padding throughout

5. Color Scheme

The application uses a sophisticated dark color palette: - **Background**: Dark gradient from #0F0F0F to #1A1A1A - **Deck Colors**: Different gray shades for deck identification - **Accent Colors**: Blue (#3498DB), Red (#E74C3C), Green (#27AE60) - **Text Colors**: White and light gray for optimal contrast

6. Animation System

The UI includes smooth animations for professional feel: - **Vinyl Rotation**: Disc rotates based on playback speed - **Beat Visualization**: Pulsing animations synchronized with beat - **Element Rotation**: Decorative elements rotate around the disc - **Smooth Transitions**: 33 FPS animation rate for fluid motion

---

R4. Technical Research and New Feature Details

Beat Grid System - Advanced Feature

1. Technical Background

The Beat Grid system is inspired by professional DJ software like Serato DJ Pro and Native Instruments Traktor. This feature provides automatic BPM detection and beat synchronization, which is essential for professional DJ mixing.

2. BPM Detection Algorithm

The system implements multiple BPM detection methods:

Energy-Based Detection:

```
double calculateEnergy(const AudioBuffer<float>& buffer) {
    double energy = 0.0;
    for (int channel = 0; channel < buffer.getNumChannels(); ++channel) {
        const float* channelData = buffer.getReadPointer(channel);
        for (int sample = 0; sample < buffer.getNumSamples(); ++sample) {
            energy += channelData[sample] * channelData[sample];
        }
    }
    return energy / buffer.getNumSamples();
}
```

Tap Tempo Detection:

```
void calculateBPMFromTaps() {
    if (tapTimes.size() >= 2) {
        double totalInterval = 0.0;
        for (size_t i = 1; i < tapTimes.size(); ++i) {
            totalInterval += tapTimes[i] - tapTimes[i-1];
        }
        double averageInterval = totalInterval / (tapTimes.size() - 1);
        currentBPM = 60.0 / averageInterval;
    }
}
```

3. Beat Grid Visualization

The system provides real-time beat grid visualization:

Grid Drawing:

```
void drawBeatGrid(Graphics& g, Rectangle<float> bounds) {
    double beatInterval = 60.0 / currentBPM;
```

```
    int numBeats = static_cast<int>(trackLength / beatInterval);

    for (int beat = 0; beat <= numBeats; ++beat) {
        float x = bounds.getX() + (beat * beatInterval / trackLength) * bounds
        g.setColour(gridColor);
        g.drawVerticalLine(static_cast<int>(x), bounds.getY(), bounds.getBotto
    }
}
```

4. Grid Synchronization

The beat grid provides automatic synchronization features: - **Snap-to-Grid**: Track positions automatically snap to beat boundaries - **BPM Matching**: Automatic BPM detection for loaded tracks - **Beat Markers**: Visual indicators for beat positions - **Sync Function**: Synchronize both decks to the same BPM

5. Technical Implementation Details

Audio Analysis: - **Buffer Processing**: Real-time analysis of audio buffers - **Frequency Analysis**: FFT-based frequency domain analysis - **Peak Detection**: Automatic detection of beat peaks - **Tempo Calculation**: Statistical analysis of beat intervals

Performance Optimization: - **Efficient Algorithms**: Optimized for real-time processing - **Memory Management**: Minimal memory footprint - **Thread Safety**: Thread-safe implementation for audio processing - **Caching**: Cached calculations for improved performance

6. Research Sources and Technical References

Professional DJ Software Analysis: - **Serato DJ Pro**: Analyzed beat grid implementation and BPM detection - **Traktor Pro**: Studied tempo detection algorithms and grid visualization - **Rekordbox**: Researched beat analysis and synchronization methods

Academic Research: - **Audio Signal Processing**: Beat detection algorithms and frequency analysis - **Music Information Retrieval**: BPM estimation techniques - **Real-time Audio Processing**: Low-latency audio analysis methods

Technical Standards: - **Audio File Formats**: WAV, MP3, AIFF specifications - **Digital Audio Processing**: Sample rate conversion and audio filtering - **UI/UX Design**: Professional audio software interface guidelines

7. Innovation and Original Contributions

Custom Features: - **Hybrid BPM Detection**: Combination of energy-based and tap tempo methods - **Visual Beat Feedback**: Real-time visual feedback synchronized with audio - **Adaptive Grid Sensitivity**: Dynamic sensitivity adjustment based on audio characteristics - **Professional DJ Workflow**: Optimized interface for professional DJ use

Technical Improvements: - **Enhanced Accuracy**: Improved BPM detection accuracy through multiple

algorithms - **Real-time Performance**: Optimized for low-latency real-time processing - **User Experience**: Intuitive interface design for professional DJ workflow - **Cross-platform Compatibility**: Consistent performance across different operating systems

---

Conclusion

The OtoDecks DJ application successfully implements a comprehensive DJ system with professional-grade features. The modular architecture, persistent music library, custom UI design, and advanced beat grid system demonstrate strong object-oriented programming principles and technical innovation.

The application provides a solid foundation for professional DJ use while maintaining extensibility for future enhancements. The combination of technical sophistication and user-friendly design makes it a valuable tool for both learning and professional applications.

Technical Specifications: - **Framework**: JUCE (C++) - **Audio Processing**: Real-time, low-latency - **File Formats**: WAV, MP3, AIFF, FLAC, OGG - **Platform Support**: Windows, macOS, Linux - **UI Framework**: Custom JUCE components - **Audio Engine**: JUCE AudioSource architecture